The Sink Trait: Code Companion

Reference code for the The Sink Trait lecture. Sections correspond to the lecture document.

Section 1: Push vs Pull Iteration Models

```
/// A trait that defines how results from searchers are handled.
///
/// In this crate, a searcher follows the "push" model. What that means is that
/// the searcher drives execution, and pushes results back to the caller. This
/// is in contrast to a "pull" model where the caller drives execution and
/// takes results as they need them. These are also known as "internal" and
/// "external" iteration strategies, respectively.
///
/// For a variety of reasons, including the complexity of the searcher
/// implementation, this crate chooses the "push" or "internal" model of
/// execution. Thus, in order to act on search results, callers must provide
/// an implementation of this trait to a searcher, and the searcher is then
/// responsible for calling the methods on this trait.
```

The documentation explicitly acknowledges the architectural trade-off. The "push" model keeps complex state management inside the searcher rather than exposing it through iterator suspension points.

Section 2: The SinkError Trait and Error Flexibility

```rust
/// A trait that describes errors that can be reported by searchers and
/// implementations of `Sink`.
pub trait SinkError: Sized {
    /// A constructor for converting any value that satisfies the
    /// `std::fmt::Display` trait into an error.
    fn error_message<T: std::fmt::Display>(message: T) -> Self;

    /// A constructor for converting I/O errors that occur while searching.
    /// By default, this is implemented via the `error_message` constructor.
    fn error_io(err: io::Error) -> Self {
        Self::error_message(err)  // Default: convert to string
    }

    /// A constructor for converting configuration errors.
    /// By default, this is implemented via the `error_message` constructor.
    fn error_config(err: ConfigError) -> Self {
        Self::error_message(err)  // Default: convert to string
    }
}

/// An `std::io::Error` can be used as an error for `Sink` implementations.
impl SinkError for io::Error {
    fn error_message<T: std::fmt::Display>(message: T) -> io::Error {
        io::Error::new(io::ErrorKind::Other, message.to_string())
    }

    fn error_io(err: io::Error) -> io::Error {
        err  // Optimization: no conversion needed for io::Error!
    }
}

/// A `Box<dyn std::error::Error>` can also be used - maximum flexibility.
impl SinkError for Box<dyn std::error::Error> {
    fn error_message<T: std::fmt::Display>(
        message: T,
    ) -> Box<dyn std::error::Error> {
        Box::<dyn std::error::Error>::from(message.to_string())
    }
}
```

The `Sized` bound on `SinkError` is required because the trait returns `Self`. The `io::Error` implementation demonstrates the optimization pattern—`error_io` avoids unnecessary string conversion.

Section 3: The Core Sink Trait

```rust
pub trait Sink {
    /// The type of an error that should be reported by a searcher.
    /// Errors of this type are used both by Sink methods AND by the
    /// searcher implementation itself (e.g., for I/O errors).
    type Error: SinkError;  // Associated type with trait bound

    /// Called whenever a match is found.
    /// Returns: Ok(true) = continue, Ok(false) = stop gracefully, Err = abort
    fn matched(
        &mut self,
        _searcher: &Searcher,       // Access to searcher configuration
        _mat: &SinkMatch<'_>,       // Match details with borrowed data
    ) -> Result<bool, Self::Error>;

    /// Called for context lines. Default: ignore and continue.
    #[inline]
    fn context(
        &mut self,
        _searcher: &Searcher,
        _context: &SinkContext<'_>,
    ) -> Result<bool, Self::Error> {
        Ok(true)  // Default implementation
    }

    // ... additional methods with similar patterns
}
```

The tri-state return type `Result<bool, Self::Error>` appears throughout the trait, providing fine-grained control over search flow. The `&Searcher` parameter lets sinks query configuration without storing it themselves.

Section 4: Lifecycle and Event Methods

```rust
/// Called when a search has begun, before any search is executed.
#[inline]
fn begin(&mut self, _searcher: &Searcher) -> Result<bool, Self::Error> {
    Ok(true)  // Default: do nothing, continue
}

/// Called when a search has completed successfully.
/// Note: returns Result<(), ...> - no bool needed, search is done
#[inline]
fn finish(
    &mut self,
    _searcher: &Searcher,
    _: &SinkFinish,  // Summary statistics
) -> Result<(), Self::Error> {
    Ok(())
}

/// Called when a break in contextual lines is found (the "--" separator).
#[inline]
fn context_break(
    &mut self,
    _searcher: &Searcher,
) -> Result<bool, Self::Error> {
    Ok(true)
}

/// Called when binary data is found during search.
#[inline]
fn binary_data(
    &mut self,
    _searcher: &Searcher,
    _binary_byte_offset: u64,  // Absolute position of binary data
) -> Result<bool, Self::Error> {
    Ok(true)
}
```

All lifecycle methods except `finish` return `Result<bool, ...>` for consistent early-termination control. The `#[inline]` hints help the compiler optimize away empty default implementations.

Section 5: Blanket Implementations for Indirection

```rust
/// Allows passing `&mut sink` instead of moving ownership
impl<'a, S: Sink> Sink for &'a mut S {
    type Error = S::Error;

    #[inline]
    fn matched(
        &mut self,
        searcher: &Searcher,
        mat: &SinkMatch<'_>,
    ) -> Result<bool, S::Error> {
        (**self).matched(searcher, mat)  // Deref twice: &mut &mut S -> S
    }

    // ... other methods follow same pattern
}

/// Enables dynamic dispatch with trait objects
impl<S: Sink + ?Sized> Sink for Box<S> {
    //         ^^^^^^^ Allows Box<dyn Sink<Error = E>>
    type Error = S::Error;

    #[inline]
    fn matched(
        &mut self,
        searcher: &Searcher,
        mat: &SinkMatch<'_>,
    ) -> Result<bool, S::Error> {
        (**self).matched(searcher, mat)  // Deref Box to inner type
    }

    // ... other methods follow same pattern
}
```

The `?Sized` bound is crucial—without it, `Box<dyn Sink>` wouldn't work because trait objects are unsized. These implementations let you choose between borrowed access and dynamic dispatch.

Section 6: Data Types - SinkFinish and SinkMatch

```rust
/// Summary data reported at the end of a search.
#[derive(Clone, Debug)]
pub struct SinkFinish {
    pub(crate) byte_count: u64,              // Total bytes searched
    pub(crate) binary_byte_offset: Option<u64>, // First binary data location
}

impl SinkFinish {
    #[inline]
    pub fn byte_count(&self) -> u64 {
        self.byte_count
    }

    #[inline]
    pub fn binary_byte_offset(&self) -> Option<u64> {
        self.binary_byte_offset
    }
}

/// A type that describes a match reported by a searcher.
#[derive(Clone, Debug)]
pub struct SinkMatch<'b> {
    pub(crate) line_term: LineTerminator,
    pub(crate) bytes: &'b [u8],           // The matched bytes
    pub(crate) absolute_byte_offset: u64,  // Position in entire input
    pub(crate) line_number: Option<u64>,   // Only if line counting enabled
    pub(crate) buffer: &'b [u8],           // Surrounding buffer context
    pub(crate) bytes_range_in_buffer: std::ops::Range<usize>,
}

impl<'b> SinkMatch<'b> {
    /// Return an iterator over lines in this match (may be multiple in multi-
line mode)
    #[inline]
    pub fn lines(&self) -> LineIter<'b> {
        LineIter::new(self.line_term.as_byte(), self.bytes)
    }
}
```

Fields are `pub(crate)` with public accessor methods—the struct can only be constructed by the searcher, but users can read all fields. The `'b` lifetime ties match data to the underlying buffer.

Section 7: Convenience Sinks Module

```
pub mod sinks {
    /// A sink that provides line numbers and matches as strings.
    /// Returns error on invalid UTF-8 or if line numbers not enabled.
    #[derive(Clone, Debug)]
    pub struct UTF8<F>(pub F)  // Tuple struct wrapping a closure
    where
        F: FnMut(u64, &str) -> Result<bool, io::Error>;

    impl<F> Sink for UTF8<F>
    where
        F: FnMut(u64, &str) -> Result<bool, io::Error>,
    {
        type Error = io::Error;

        fn matched(
            &mut self,
            _searcher: &Searcher,
            mat: &SinkMatch<'_>,
        ) -> Result<bool, io::Error> {
            // Validate UTF-8
            let matched = match std::str::from_utf8(mat.bytes()) {
                Ok(matched) => matched,
                Err(err) => return Err(io::Error::error_message(err)),
            };
            // Require line numbers
            let line_number = match mat.line_number() {
                Some(line_number) => line_number,
                None => {
                    let msg = "line numbers not enabled";
                    return Err(io::Error::error_message(msg));
                }
            };
            (self.0)(line_number, &matched)  // Call the wrapped closure
        }
    }

    /// Like UTF8, but replaces invalid UTF-8 with replacement characters.
    pub struct Lossy<F>(pub F)
    where
        F: FnMut(u64, &str) -> Result<bool, io::Error>;
}
```

The `UTF8` and `Lossy` sinks demonstrate the trade-off between strictness and convenience. These are tuple structs wrapping closures, providing a functional style for simple use cases.

Quick Reference

Sink Method Return Semantics

Return Value	Meaning
`Ok(true)`	Continue searching
`Ok(false)`	Stop gracefully, still call `finish`
`Err(e)`	Abort immediately, do NOT call `finish`

Sink Lifecycle

```
begin() → [matched() | context() | context_break() | binary_data()]* → finish()
```

Key Types

```rust
trait SinkError: Sized {
    fn error_message<T: Display>(message: T) -> Self;
    fn error_io(err: io::Error) -> Self { ... }
    fn error_config(err: ConfigError) -> Self { ... }
}

trait Sink {
    type Error: SinkError;
    fn matched(&mut self, &Searcher, &SinkMatch) -> Result<bool, Self::Error>;
    // Optional: context, context_break, binary_data, begin, finish
}
```

Convenience Sinks

Type	Closure Signature	Behavior
`UTF8<F>`	`FnMut(u64, &str) -> Result<bool, io::Error>`	Error on invalid UTF-8
`Lossy<F>`	`FnMut(u64, &str) -> Result<bool, io::Error>`	Replace invalid UTF-8

`Bytes<F>`	`FnMut(u64, &[u8]) -> Result<bool, io::Error>`	Raw bytes, no conversion